APPENDIX

VERSION OF PARAGRAPHS OF THE SPECIFICATION WITH MARKINGS

Page 2, Paragraph 1:

CROSS REFERENCES TO RELATED APPLICATIONS

The present application is related to**, and claims priority from,** co-pending U.S. patent application 60/206,564 entitled "METHOD AND SYSTEM FOR MANAGING PARTITIONED DATA RESOURCES," and filed on May 22, 2000, currently pending. The **[Aforementioned] above identified** application is incorporated by reference in its entirety.

Page 135, Paragraph 2:

At runtime, the use of the logical domains is particularly relevant in the entity creator methods and the entity finder methods. It can be understood from the description of creating an entity as described with respect to **FIG. 21** above, that there is no explicit notion of *where* to create it; the "where" question is answered implicitly by the entity container that the client has an interface to. With respect to the present invention, creator methods are introduced that allow the specification of *where* to create the instance. Each entity's create interface (implemented by the satellites) needs to supply a createInPartition() method that explicitly indicates the physical partition in which the new instance should be created. Also required is a createInDomain(String domain) method that allows the user to specify in which domain the instance should be created. This method would first use directory services to map the specified domain name onto the set of partitions that belonged to that domain. It would then use some policy (*e.g.*, [.] random selection or greatest available capacity) to select one physical partition from the set of qualifying partitions; the new instance would then be created in that partition. In accordance with a further embodiment of the present invention, a createInDomains(String[] domains) method searches the directory for all partitions belonging to all specified domains (intersection) to deploy a new entity instance in a partition that concurrently belonged to two or more domains (e.g., placing an entity simultaneously in Europe and Internet LOB domains).

Page 135, Paragraph 3:

In accordance with another embodiment of the present invention, another major area where domains would be visible in the user interfaces is in the complex finders for an entity. These find-by-criteria methods are given an extra argument for naming one or more domains to be intersected. The find operation is then performed in parallel out at all partitions matching the specified domain(s). The results of the parallel queries would then be coalesced and returned to the requestor.

Page 139, Paragraph 1:

With reference to **FIG. 29**, a diagram of NW service platform infrastructure of interrelated services relating to an enterprise is illustrated in accordance with an exemplary embodiment of the present invention. There, entity servers **2902A** and **2902B** are shown with the respective databases **2904A** and **2906A** for server **2902A,** while databases **2904B** and **2906B** are hosted by server **2902B**. In the depicted Figure, each server has two VM containers **2908** and **2910** running, and each container has two NW partition services running within. Partitions **2908** and **2910** are responsible for two main things -- retrieving one or more instances of a business object and creating a new instance of a business object. Typically, client **2940** would not directly invoke methods of the partition, but would utilize instead a satellite service. Notice that the figure depicts four entity classes, **A - D**, representative of, for example, Customer, Account, Billing Address and Pending Order entity classes. Notice also that each of the entity classes is partitioned. With respect to the present Figure, each entity has two partitions, but in practice, most entities would have many more partitions. Each partition is responsible for a plurality of entity instances which are identifiable by a primary key. Also depicted is registrar **2930** which may be a domain registrar as described above with respect to **FIG. 9**. It is expected that the business objects normally used by a client are proximate to that client, thus a fair assumption is that all components represented in **FIG. 29** are in a local domain, such as the local domains defined by a multicast radius as further described above with respect to **FIG. 9**. However, as has been alluded to above, and which will be described in greater detail below, a client may interact with business objects located anywhere in the enterprise, locally or non-locally. Thus, servers **2902A** and **2902B** may or may not be local, while registrar **2930** and finder **2932** are local to client **2940**. However, the operations that each of these services perform might lead to hops in other non-local**[,]** domains.

Page 146, Paragraph 2:

FIG. 33, on the other hand, is a flowchart depicting a process for getting all accounts instances that are associated with an identified customer instance in accordance with an exemplary embodiment of the present invention. It is assumed that the client has already found an association service and has association proxy **3230** to interface with the service. The process begins by identifying the partition container holding the entity instance (step **3302**). Although this might be accomplished via finder service **3232**, as described above, this extra lookup is not necessary. The smart proxy that serves as a remote handle to an entity actually encapsulates a remote reference (typically an RMI stub) to the entity partition container, as well as other information like PK. In either case, the partition container must be found for the entity instance in order to locate the association engine fragment that is coupled to it (step **3304**). Once the container is found, the interface to the coupled association engine fragment is gotten (typically via local registrar service lookup) and the Customer/Account association engine fragment traversed starting from the Customer instance to find all associated Account instances (step **3306**). Those instances are then returned to the association engine (step **3308**) which passes them on to client **3240** (step **3310**). At step **3308,** it is assumed that the remote interfaces are returned for the Account instances and the client interacts with the Account instances as need be. In one embodiment of the present invention, the link records held in the association fragment engines consist of the triplets (primary key, entity type, partition number) for both source entity and destination entity that are linked. In traversing the association, the association fragment engine must query its link record store for all link records matching a given source entity. Then, given a set of link records, it must resolve the PK, entity type and partition number for destination entities into remote smart proxies for those entities. **T[t]his** could be done via the multi-hop find-by-PK. In practice, we optimize this by caching, in the association fragment engine, a map from (entity-type, partition #) onto the remote RMI stub to the corresponding entity partitions. Should this stub lookup suffer a cache "miss", the interface to the partition

service is fetched via enterprise service lookup, described above, via a partition naming convention formed from the entity type and partition number. In either case, once remote reference to destination partition or partitions is available, the association fragment engine can request those partitions to return smart proxies corresponding to all the destination entities matching the destination PKs.

Page 148, Paragraph 2:

One aspect of one embodiment of the association approach of the current invention is the novel use of smart proxies in the interfaces to the logical association engine services. In this embodiment, the association engines are accessed through a service interface that is implemented with a smart proxy that, itself, contains no inherent remote references (proxies in stubs) to a remote object. **T[t]he** association engine smart proxies "piggyback" on the communications channels of the entity smart proxies that are passed to it as parameters in "link()" or "traverseAssociation()" requests. For example, when the association interface is told to "link" entities A and B, the association smart proxy will extract from A and B their remote handles to their respective partition containers. The smart proxy will then proceed to send parallel link requests (in separate threads) to these two entity containers, who, in turn, forward the requests to the appropriate association fragments. This is a highly novel and atypical example of the use of a smart proxy. It has no communications "channels" of its own, but rather "parasitically" employs the communications channels of objects with which it comes into contact.

Page 149, Paragraph 1:

Another feature of one embodiment of the present invention is the manner in which the virtual association engine deals with violations of cardinality integrity. Specifically, when a client requests that a one-to-many association add a new link record, the system must check for cardinality violations. Due to the asymmetric nature of a one-to-many association and the DataBus approach of partitioning entities, only the fragment engine on the "many" side can reliably detect a cardinality violation. A simple minded implementation would serialize the link requests to the association fragments on either side of the relation, first to the "many" side, then only if successful, sending the link request to the "one" side. In one embodiment of the current invention, we forward these link requests in parallel (using two background threads) to the two "sides" of the association. If the "many" side detects a cardinality violation, it will throw an exception. **T[t]he** "one" side will, in any case, proceed to add a link record, all-the-while ignorant of whether doing so violates cardinality constraints. But all such link operations are performed under the oversight of a global transaction. Thus, when the caller of the (illegal) link request catches the cardinality exception, they will (according to the "contract" they are expected to obey) "roll back" the global transaction. The "one" side's inappropriate adding of an illegal link record will effectively be undone.

Page 149, Paragraph 2:

Management Operations Center Overview

The Management Operations Center (MOC) is an application for providing support for people addressing problems similar to those handled in a Network Operations Center (NOC), but not limited to only network problems. As such, it is intended to support problem management in many forms, including those typically handled by customer support centers[,] and tactical assistance centers. The MOC represents a tool that assumes a fundamental re-engineering of the processes and tools used in these environments. It should not be compared directly against the tools that currently support these environments, but analyzed as to how it supports the new re-engineered process. As such, it will not support many things currently expected in these environments[,] because some activities are not needed.

Page 150, Paragraph 1:

The current NOC environment can be described in simple terms as an approach involving monitoring of activity, identification of problems, selecting of problems to work on off of a queue, and resolving the problem. By contrast, in accordance with an exemplary embodiment of the present invention, the MOC monitors and identifies problems based on rules set up by experts. Additionally, rather than an *ad hoc* **personnel [personal]** deployment, the MOC**[;]** determines the best available **personnel [personal]** for a particular problem based on rules and then directly invites those persons to work on the problem. Therefore, **[to and]** the work team is composed based on differing roles and skill sets required for the problem, and might involve people from different organizations. Because the MOC is an integration of services, the MOC is able to handle problem cases that are not limited to one area, as is the practice of Network Operations, but to any areas affected. For instance, a problem may bind together a network event, customer tickets, application events, etc. Finally, in accordance with an exemplary embodiment of the present invention, a work event can be worked on and accessed by anyone with connectivity to the NewWave environment, so people involved do not have to be in one center, but could be at home, on the other side of the world, etc. Thus, in stark contrast with prior art attempts, the MOC's emphasis is on collaboration tools and world-wide access.

Page 150, Paragraph 2:

Operations support systems today tend to be large, closed applications that perform part of the work needed by OSS personnel. OSS personnel usually end up using several systems that overlap and do not talk to each other. As opposed to a closed application that provides merely a partial solution, the MOC of the present invention represents an example of a new way of designing applications: the inside-out design. In this approach to building systems, rather than building monolithic application systems, the "application" is a collaboration of many smaller units acting on common objects, possibly without knowledge of each other, but with their actions affecting each other. This design also makes heavy use of rules external to code executed by rules engines. This allows for **[the]** changing **[of]** the behavior of the system without changing the code. Those behaviors which represent organizational policy are removed into rules, and can then be managed by experts in those organizations.

Page 151, Paragraph 2;

Aggregator **3506** receives event information from pub/sub bus **3528** and assessor **3504** which it associates and binds together according to **an** operation's requested organization of work integration to produce a work document. Aggregator **3506** also provides real-time binding of associated corporate business objects to the document including binding an event to many different documents. To that end, aggregator **3506** contains the templates for documents for, for example, different functional areas/teams. Additionally, many different aggregators will exist and run simultaneously providing different work documents to different teams. Dispatcher **3508** applies current policy rules to associate work documents and events with specific operators, customer contacts and other service care staff. Dispatcher **3508** assigns work with an understanding of who is free and able to do that work and implements priority rules for **understanding [understands]** relative priority, thus dispatcher **3508** can bump work in progress for higher priority tasks. Additionally, dispatcher **3508** implement alternate strategies to handle cases where work is refused or overdue. Distributor **3510** handles outbound notifications for the MOC based on decisions from dispatcher **3508**.

Page 152, Paragraph 1:

Each avatar object **3512** represents a virtual image of what a specific operator or customer contact is skilled at and responsible for. Operators, provisions, customer contacts, service support staff and any other management-tasked staff in the customer and network care environment **[and]** will have an avatar. Avatar **3512** provides the MOC with a skills assessment of care staff including reference to a history of past work, interactions and success ratings.

Page 152, Paragraph 2:

Archive service **3514** updates and otherwise modifies work documents for or in storage based on recent experiences. Finally, work rendezvous **3516** associated later arriving processed events with events which initially generated a work stream/task. With this service, different people working on the same route or associated task can learn of the complementary work going on. Rendezvous **3516** notifies different work documents about all other references to a common event and associates processed events with a work document that contain a reference to the source event. The GIB services have been discussed thoroughly above so will not be discussed again with respect to **FIG. 35**.

Page 152, Paragraph 3:

The key features of the MOC design are:

Rather than a single monolithic application, the MOC employs an *inside-out* **design** in which many small components act largely independently of each other, but affecting each other by:

directly interacting with shared resources;

registering for notification of updates to shared resources;

finding each other and communicating via the GIB services of registration and lookup; **and**

publishing messages over the GIB publish/subscribe bus.

Page 153, Paragraph 1:

In general, the overall behavior of the MOC can be changed by adding new components, without directly modifying existing components. All MOC components are NewWave services, using the NewWave registration, lookup and enterprise lookup services.[;] The MOC extends the behavior of its code through the use of external rules engines using the NewWave behavior service. This allows organizations with the expert knowledge of operations support to be in control of the behaviors implementing operations support policy, instead of programming organizations.[;] The MOC must communicate with systems outside of its direct control. It uses XML messages sent over the GIB publish/subscribe buses to do so in a highly decoupled way. In this way it uses a common approach for receiving events from disparate sources: external sources such as the network, customer service systems and legacy systems; and internal sources such as MOC or NewWave components.[;] Events received are assessed by an assessing component. It is the job of this assessor to apply organizational policy dealing with *how* a event should be addressed.[;]

Page 154, Paragraph 1:

Work documents are constructed by an aggregating component. It is the job of this aggregating component to put together a document containing *what* information is needed to resolve the problem, including aggregating related network, customer and application events, setting initial milestones, applying rules which deal with the understanding or categorization of the problem, and binding in related information (such as necessary topology information).[;] Work documents are first class objects[,] accessible through the NewWave DataBus. They are not just data, as they have a controlling feature that allows state changes throughout its life. All components **are** able to interact with and change it using distributed transaction semantics.[;]

Page 154, Paragraph 2:

People are invited to participate in work groups by a dispatching component running dispatching rules. It is the job of this dispatcher to apply organization policy dealing with *who* should deal with a problem.**[; and]**

Page 154, Paragraph 3:

People are represented by in-memory "avatars" which are responsible for knowing the manner in which to communicate with the person, that person's current workload and information about the person. Technologies for communication are encapsulated within the avatar, allowing other components to not be knowledgeable about or bound to those technologies. An avatar is a first-class object running as a service**[,]** and can be found and interacted with using standard NewWave techniques.

Page 154, Paragraph 5:

FIG. 36 is a functional diagram of the MOC depicting interactions between key MOC components **which** interact in accordance with an exemplary embodiment of the present invention. Note that **FIG. 36** contains component representations from the MOC, NewWave infrastructure and rule engines **3630**. The MOC is responsible for coordinating the response to events which occur within components **3640** developed for the NewWave environment. Components **3640** might be NewWave-enabled devices, and applications and services developed on NewWave or are detected by NewWave surrogates for external systems, for instance, agents monitoring devices or other resources, and bridges to legacy (non-NewWave) systems. None of these components and surrogates **have[has]** any special knowledge of the MOC or its functionality and thus are not modified in any special way to interact with the MOC. These components and surrogates are responsible only for knowing when a problem has occurred, and **for** publishing an XML document describing the problem on the publish/subscribe bus. The only coordination which occurs between the MOC and components **3640** is in the definition of the XML document and the topic that it is published with **it**.

Page 155, Paragraph 1:

To make any component able to be supported by the MOC, it must be able to interact with the MOC components, **even[event]** though components **3640** **do[don]** not have any specific knowledge of the MOC components themselves. This is accomplished by using an agreed upon format and transmission media. To that end, any component wishing to be supported by the MOC must publish events to the GIB publish/subscribe bus, or to an external publish/subscribe bus with a bridge to the GIB, shown generally as pub/sub bus **3618**. Each event message must use an agreed-upon XML schema for the format and follow an agreed-upon structure for the topic of the event message.

Page 155, Paragraph 2:

The precise XML document standards are not **important[import]** for the purposes herein, but some exemplary events are listed below.

FMEvent XML (Fault Management Event) XML document, topic ngn.nfp.fm

Application Event XML document, topic ngn.service.admin

Customer Service Event XML document, topic ngn.customerservice

UUNet Ping Alarm FMEvent XML document, topic ngn.uunet.fm

Page 156, Paragraph 2:

Simply put, assessor **3604** runs a rules engine against an incoming event received from publish/subscribe bus **3618**. In accordance with an exemplary embodiment of the present invention, assessor **3604** determines, based on the rules, whether the event is a *primary* event, which must be investigated and then classify the event by type. The function of assessor **3604[,]** then**[,]**, is entirely dependent upon the rules that are set up and executed by the rules engine. These rules would be defined by experts in an organization or set of organizations responsible for handling problem events. An exemplary rules engine for implementing policy based rules is Brokat Advisor and the Brokat Advisor Server (both available form Brokat Aktiengesellschaft Industriestrasse 3, D-70565 Stuttgart, Germany). Because the present invention is supported by the NewWave infrastructure, which in some forms **rely[relies]** on the Java programming language, a rule engine that is also written in Java would better matched the operating environment of the MOC.

Page 156, Paragraph 3:

Referring to **FIG. 37**, an assessor **is depicted** for assessing events based on organizational rules in accordance with an exemplary embodiment of the present invention. Notice that **[to]** the present invention envisions implementing rules in the normal manner by developers in development **3710**. However, the present invention recognizes that the developers are not always the best implementers for rules, especially those based on organizational policy, such as from operations **3712**. Therefore, the rules may be adjusted, modified, supplemented or even replaced by experts in an organization or set of organizations responsible for handling problem events at, for instance, code server **3720**. Code server **3720** then serves up the rules to rules agents in behavior service **3730**, which are fire whenever an event is received.

Page 157, Paragraph 1:

Work Item Aggregation

The next step involves building the case, involving the separate activities of:

1. the correlation of the primary event and related events into a single bundle;
2. the building of a work document containing the bundled events and the other related information;
3. the running of rules designed to help determine the cause of the problem; **and**
4. the control of the life of the work to be done.

Page 157, Paragraph 3:

Many separate aggregators might be deployed **[in the]** within the MOC to and the point at which it became involved could be modified. For instance, it is entirely reasonable for an aggregator to begin work before the assessor, bringing the case up to a certain point to give the assessor adequate information to make its assessment. Then, it could continue its work conditional on the assessment.

Page 157, Paragraph 4:

The basic design of an aggregator is shown in **FIG. 38** in accordance with an exemplary embodiment of the present invention. **FIG. 38** also describes the basic workings of aggregator **3706** in accordance with the present invention. Accordingly, events flow into aggregator **3806** and pass through a "gate,"**[,],** logic gate **3740**, which applies some logic to determine whether this event represents a new or existing opportunity for work. Logic gate **3740** acts as a logical IF to determine if the event represents new or existing opportunities.

Page 158, Paragraph 1:

If new, aggregator **3706** starts a new state machine, depicted as state machine **3850**, for controlling the work. It must select a template for the state machine that is appropriate for the particular event. For instance, a failure of a Sonet Ring would be handled differently than a failure of a modem. State machine **3850** performs a number for functions regarding the event including determining other related events **needing[need]** to be bundled with the primary event and what events would signify closure of the primary event. Additionally, the state machine might need additional information so the state machine must be equipped to determine the additional information that is needed, such as topology or customer service level agreement (SLA). Finally, the state machine **3850** must determine what milestones in the life of a particular event are important.

Page 158, Paragraph 3:

[FIG. 39 is a diagram of a simplified version of a state machine in accordance with an exemplary embodiment of the present invention.] **FIG. 39** is a diagram of a simplified version of a state machine in accordance with an exemplary embodiment of the present invention. State machine **3840** does the actual work in the aggregator.

Page 159, Paragraph 1:

The aggregator may be implemented in a number of different ways or depending on the type of state machine selected for use, **although it** might be omitted from the MOC altogether, *i.e.*, in accordance with one embodiment of the present invention, there is no physical component called the aggregator. In that case, state machine **3940** subscribes with the publish/subscribe bus for the events **in which** it is interested**[in]**. As such, state machine **3940** receives the events directly without going through a "gate," thus performing the join function implicitly. Additionally, the MOC rendezvous service, discussed below, is used to prevent an event which has gone directly to state machine **3940** from also causing the creation of a new state machine, thereby also performing the gate function. Finally, the assessor is allowed to create the state machine if the event is determined to be primary and the rendezvous service says it is not being handled already, thus performing the start function of the aggregator.

Page 159, Paragraph 3:

One exemplary state machine **[it]**is implemented as an extension of the base class WorkItemActor. The particulars of this exemplary implementation are that the publication of the document is done through publish/subscribe and after publishing the document, the state machine does not go away. Instead of disappearing, the state machine continues operating for the life of the work. This is necessary because even after the document is produced, related events will still keep coming in and need to be joined to the existing work document. Rather than create a new component for doing this function after the work document was created, the state machine was allowed to continue to live on its own.

Page 160, Paragraph 2:

Event and Work Item Rendezvous

As mentioned above, the MOC depends on a rendezvous service to tell if an event is being handled by a work item (or a state machine controlling a work item), **represent[represented]** in **FIG. 36** as rendezvous **3616**. A rendezvous service is responsible for determining if a given event **[an]** is already being handled and **[determining]** the problem represented by a work item document overlaps with an existing work item document.

Page 160, Paragraph 3:

As with many other MOC components, many rendezvous services can be deployed simultaneously in a MOC environment. Rendezvous service **3616** can use rules to make its determinations, or any logic appropriate. Different rendezvous services may be deployed with different rules for determining overlap by different parameters. For instance, different services could consult different views of topology, for instance, one service could look at layer 2, a second a layer 3, a third looking **at only[only at]** cross domain interactions. By deploying many such rendezvous services, and by allowing work documents already started to be merged together, it is not necessary to apply all rendezvous rules before starting work. Therefore, some rendezvous rules could be quite slow and still result in alerting people to the fact that a problem spans multiple areas.

Page 161, Paragraph 1:

In accordance with an uncomplicated implementation of rendezvous service **3616**, it applies simple rules to determine if an event has been handled and if there is overlap between different work item documents. Rendezvous service **3616** subscribes with publish/subscribe to be made aware of any new work item document as the item is created (but before it is published for all). Rendezvous service **3616** pulls the events off of the document and keeps an index of events based on their type and **[based]**on the network element affected. Whenever it is asked about a new event, it can compare the new event against the network element to see if any existing work item document references that element.

Page 161, Paragraph 2:

The Work Document

WorkItem **3630** in **FIG. 36** holds all of the objects associated with the item of work, i.e., the problem described in the initial event that is being addressed. In accordance with an exemplary embodiment of the present invention, WorkItem **3630** will **[be]**have many types of objects, bound in, these include:

events - the primary event and correlated events**[ ]**;

status information describing the current state of the problem resolution, for instance, status, probable cause, priority, time to resolve, actual cause;

informational objects, such as:

customer information and service level agreement information;

topology information;

basic instructions regarding the problem;

progress notes; and

sundry information such as configuration information on the elements in question;

user avatars for the people participating in the work group;

proxies to devices or services related to the problem, for instance, a proxy to the device or agent for the device that is reporting the problem; and

tools needed to resolve the problem, including collaboration tools for interacting with the work group, or getting information about the problem elements.

Page 162, Paragraph 2:

Similar to WorkItem **3630**, the MOC design has the concept of the work group space, a space for the work group collaborating on a problem to share relevant objects. This is an in-memory shared space capable of holding any objects which the participants may need, including:

the work item document itself;

proxies to user avatars for communicating with other participants;

work flow objects;

an active object controlling the lifecycle;

proxies to intelligent devices and agents for devices to interact directly with these devices;

user interfaces to access the work item information (allowing different user interfaces for different roles - technical support, customer support, **etc.); and[etc)** collaboration tools.

Page 163, Paragraph 1:

The work group space must support the putting of objects into and the getting of objects from the space, the remote downloading of the classes needed to use the objects, the registration for and notification of events relating to the objects, and the tailoring of the contents of the space to the particular problem using rules. These concepts are features of the NewWave infrastructure and GIB architecture, thus easily implemented. **Alternatively[Alternative]**, with the exception of the rules, this resembles the responsibilities of a Jini JavaSpace, which could be also used to implement the space.

Page 163, Paragraph 2

WorkItem **3630** is very similar to the work space concept of the present invention and performs most functions of the work**. This function**, however, as a DataBus object, WorkItem **3630**, is not practical to support those objects that are not really intended to be persistent. The practical answer to implementation is to convert WorkItem **3630** into an in-memory work group space with a simple interface for finding objects and retrieving them, and a persistent work item document for persisting data about the work item. In accordance with an exemplary embodiment of the present invention, there are three alternative implementations of the work group space.[,] **The first implementation[First]** is **[as]**a NewWave service, described in detail above, registering itself in the domain registrar and the enterprise repository. This implementation allows the work group to be accessed via normal administrative tools for services;**[,]** however, large numbers of work group spaces could get unwieldy. A second implementation involves creating an in-memory DataBus object. This would scale well to large numbers, but would not be directly accessible via normal service administrative methods. Finally, the work group can be implemented as a JavaSpace. This would require the **addition[additional]** of proxies so that one JavaSpace could service a number of work group spaces.

Page 164, Paragraph 1:

Creating a work group to handle the event

User Avatar Service (User Proxy)

In accordance with an exemplary embodiment of the present invention, contacts (e.g., Operators, Provisions, Customer Contacts, Service Support staff, any other management-tasked staff in the customer and network care environment, Customers, etc.) may participate in the resolution of, or may need notification of, WorkItem **3630**. As such, it is necessary to understand what the operations staff is currently working on real-time, **what the task priority is, and when it is due to be completed[what is the task priority, and when is it due to be completed]** - workload, and how to route messages and work to a contact.

Page 164, Paragraph 2:

In some cases, **in order to support the dispatching function of the MOC, the following additional information is needed[additional information is need in order to support the dispatching function of the MOC:]**:

for care staff - skills assessment;

for customer and network care staff - a reference to history of past work;

interactions, and success ratings (knowledge base);

data on **domains[domains]** of responsibility (assignments);

physical location; and

availability.

Page 164, Paragraph 3:

Contact data can then be **saved[save]** in an appropriate location, for instance, persisted within a Contact DataBus Entity.

Page 164, Paragraph 4:

In accordance with an exemplary embodiment of the present invention, a user avatar, depicted as user avatar **3604** on **FIG. 36** represents a virtual image of what a specific operator or customer contact is skilled at and/or responsible for. A user avatar is depicted as user avatar **3604** on **FIG. 36**. User avatar **3604** serves as a proxy for a contact within the MOC. Optimally, each contact has an associated user avatar. This concept is better understood with respect to **FIG. 40** which depicts a user avatar lookup in accordance with an exemplary embodiment of the present invention. **[There, contacts are]**

Page 165, Paragraph 1:

Lookup

Each UA **4012** registers in registration service **4022**, which may be a domain registrar, started with group **"Users."["Users".]** UA **4012** is registered with attributes including primary key (PK), login ID and name. Any service requesting information from, or sending messages or work to, UA **4012[,]** locates this proxy using existing NewWave protocols for lookup described above.

Page 165, Paragraph 2:

Contact DataBus Entity

UA **4012** is initially populated from the Contact DataBus Entity stored in DataBus **4024**. Once created, UA **4012** has the ability to synchronize its data with the DataBus. Additionally, UA **4012** provides "helper" convenience methods**[,]** so that data persisted with the Contact DataBus Entity can be accessed through UA **4012**.

Page 165, Paragraph 3:

Contact Means

As a proxy to the contact, the User Avatar knows all available contact means for a contact, and is responsible for forwarding all communications, messages and work[,] to the contact via the appropriate contact mean(s). All logon/logoff requests from the WorkSpace applet (GUI) will be made through UA **4012**. A remote proxy to the WorkSpace will be retained as an available contact mean after a "logon" request from the WorkSpace GUI has been successfully processed. Likewise, all logon/logoff requests from the PDA service[,] will be made through UA **4012**. Again, a proxy to the PDA Service will be retained as an available contact mean once a "logon" request from the PDA service is successfully processed. Other contact means supported by the current implementation include text messages sent to pagers via email, text messages sent to cellular phones via email, and email.

Page 166, Paragraph 3:

Subscriptions

UA **4012** has the ability to publish and subscribe via the GIB's pub/sub bus **3618**. In the MOC, UA **4012** subscribes for WorkItem status changes[,] so that UA **4012** can forward status change messages to the Contact via the available contact means. Contacts also subscribe for items of interest within the MOC via their established contact means. These subscriptions are established with UA **4012** as profile information. Once UA **4012** receives a subscribed item, it is responsible for forwarding the corresponding messages in the proper format to the Contact via the established contact mean(s).

Page 166, Paragraph 4:

Statistics

UA **4012** implements the MOC's Service Admin interface, and provides statistics as XML including health / heart beat, and MOC specific statistics such as**[,]** workload**[,]** and profile information (current subscriptions, contact means, etc.).

Page 167, Paragraph 2:

It should also be noted that the data, rules, subscriptions, and templates for the current implementation of the dispatcher service have been limited in scope to a particular telecommunications challenge business scenario. As such, this document is intended to describe this service as a participant in the Management Operations Center (MOC). No attempt has been made **[by]**to describe all data, rules, subscriptions, and templates needed to support all policy function necessary for the MOC.

Page 167, Paragraph 3:

The Dispatcher Service (GenericGIBService) provides the MOC the ability to:

1. apply current policy rules to associate work documents and events with specific operators, customer contacts and other service care staff**;[,]**
2. **assign[assigns]** work (WorkItems) with an understanding of who is free and able to do that work**;[,]**
3. **understand[understands]** relative priority and can bump work in progress for higher priority tasks**;[,]**and
4. identify Customer Contact(s) that should be notified of WorkItems based on their Service Level Agreement (SLA).

Page 168, Paragraph 1:

The Dispatcher Service accomplishes the above stated objectives by implementing the following functions:

1. Register to receive WorkItem change events via the publish/subscribe bus.

As a GenericeGIBService, the Dispatcher Service inherits the ability to participate in the publish/subscribe bus. Upon service start-up, the Dispatcher Service registers as a subscriber for WorkItem Events which include, but are not limited to:

WorkItems that have had a status change such as "opened" or "escalated";

WorkItems whose key information has changed requiring a rerun of the dispatching rules;

Invitations to participate in a WorkItem that have been "declined" by a candidate Contact; and

Invitations that have **"expired," i.e.,["expired",]** not accepted or declined by a candidate contact, and need to have an alternate contact assigned to the associated role.

Page 168, Paragraph 3:

2. Run dispatching rules.

The Dispatcher Service passes the WorkItem to the rules engine for processing. Upon receipt of the WorkItem, the dispatching rules must determine which rules to run based on the type of WorkItem Event. In most cases, the first task is for the rule engine to determine:

1. Identification of contacts that need to participate in the WorkItem

The dispatching rules are responsible for determining which contacts need to be invited to participate in, and ultimately resolve, the WorkItem. In order to define the participation needed to resolve the WorkItem, rules identify **roles** to be filled by "appropriately skilled" Contact(s). The roles are determined by rules that evaluate the WorkItem's data. Although the WorkItem contains comprehensive information, the event data currently evaluated by the dispatching rules for the purpose of determining roles includes:

the primary event type;[,]
the primary event location;[,]
the type of equipment involved;[,] and
the severity of the primary event.

Page 169, Paragraph 3:

In addition to evaluating the event data on the WorkItem, the dispatching rules evaluate customers who have been impacted by the event(s). Impacted customers and their corresponding Service Level Agreement information have been bound into the WorkItem prior to receipt of the WorkItem by the Dispatcher Service. The dispatching rules determine whether direct customer participation in the WorkItem is required. Additionally, the dispatching rules determine whether it is necessary to assign a Contact specifically to the impacted customer. The customer participation rules evaluate:

the customer's **Service Level Agreement (SLA)[SLA]** to determine whether their notification of the WorkItem is required; or

whether the customer has been located on the customer service special handling list.

Page 170, Paragraph 2:

Once the roles have been identified by the dispatching rules, a second set of rules are run to determine the "most appropriate" Contact(s) to fill the roles. Contacts can be defined as interested parties, not limited to people, that have been identified by the dispatching rules as **being** available for participation in a particular WorkItem. In order to determine the "most appropriate" Contact(s), the dispatching rules perform pattern matches on Contacts' characteristics. In the current Dispatcher Service implementation, the dispatching rules define Contact characteristics as:

skills - Does the Contact possess the appropriate skill type and level to fulfill the role?

experiences - Has this **C**ontact solved this problem or a problem like this before - history?

assignments - Is this Contact currently assigned to this Customer, Vendor, System, or piece of equipment?

physical location - Does the fulfillment of this role require physical proximity to the event location?

availability - Does the Contact's current workload allow participation in the WorkItem?

Page 171, Paragraph 1:

2. Invitation creation

As each candidate Contact is identified by the rules, a function is invoked to create an Invitation object and bind it to the WorkItem. Each Invitation contains base WorkItem information, such as event type, priority, contact's proposed role, contact's **ID[id]** and WorkItem ID. The initial invitation status is "ready for **dispatch."[dispatch".]**

Page 171, Paragraph 2

3. Instructions / Scripts bound into the WorkItem

The dispatching rules evaluate the roles, customers' SLAs, and event information to determine whether instructions or scripts need to be bound into the WorkItem for the candidate contacts. Once all contacts have been identified, invitations have been created, and instructions have been bound into the WorkItem, the dispatching rules engine returns to the Dispatcher Service for further processing.

Page 171, Paragraph 3:

3. Dispatch the Invitation to the Contact's proxy.

The Dispatcher Service queries the WorkItem for a list of invitations that need to be dispatched[,] and then performs a registrar lookup for the Contact's proxy (UserAvatar) by Contact ID. Upon return of the contact proxy from lookup, the Dispatcher Service performs a "send" request passing the invitation. The contact proxy is responsible for determining where the invitation should **be** sent and how it should be formatted.

Page 171, Paragraph 4:

Distributor Services

In accordance with an exemplary embodiment of the present invention, messages[,] in the Management Operations Center (MOC)[,] need to be distributed to contacts. Within the MOC, messages take several formats: XML, HTML, text, and direct communication with a remote proxy. The messages and may be disbursed via a variety of communication mechanisms: PDA, email (WorkSpace servlet), pager, mobile phone, and WorkSpace GUI Client (applet).

Page 172, Paragraph 2:

Distributor Service

The Distributor Service of the MOC**[,]** is responsible for distributing messages via email. The current implementation of the Distributor Service implements the Java Mail**[,]** classes and distributes messages to mobile phones, pagers, and email.

Page 173, Paragraph 1:

The archive service implementation for **[in]** the **[current]** MOC performs a simple calculation on the experience level of each person in the role played in the work group. Each time a person participates in a successful resolution of the problem, that person's experience level is modified according to the following formula: current+((max-current)*.5). This gives the person a lot of credit the first time that person is successful, but less **credit** each time **[after]** until the score is close to the maximum. .

Page 176, Paragraph 1:

METHOD AND SYSTEM FOR MANAGING PARTITIONED DATA RESOURCES

ABSTRACT OF THE DISCLOSURE

In accordance with an exemplary embodiment of the present invention, association forming entities are: a) maintained as objects in a like manner to the data objects being associated; and b) are themselves partitioned objects comprising two or more association fragments, each association fragment being mostly concerned with the interfaces to a particular data object participating in the association. In accordance with an exemplary embodiment of the present invention, each association fragment affiliated with a particular data object is stored in a location that enhances the ease of interaction between the association fragment and the data object. For example, where a first data object and second data object are maintained in data stores at some distance from one another, physically or logically, then a first association fragment will be located with or near to the first data object, and a second association fragment will be located with or near the second data object, at least within the same partition. This arrangement may be preferable because the volume of interaction between a data object and its respective association fragment may far outweigh the interaction needed between the two association fragments. This arrangement may also be preferable as the volume of interaction between a client application and both the data object and respective association fragment may exceed the interaction needed between the two association fragments. Some interactions will employ only one of the association fragments with the net result being a reduction in communications requirements and an improvement in performance. The present invention further provides for defining logical domains which are arbitrary and entirely orthogonal to partitions.